ONDERZOEKSRAPPORT NR 9223

# Enhanced Decision Modelling through a Decision Table Engineering Workbench

by

J. VANTHIENEN

M. SNOECK

D/1992/2376/29

# Enhanced Decision Modelling through a Decision Table Engineering Workbench

J. VANTHIENEN
M. SNOECK
Katholieke Universiteit Leuven
Department of Applied Economic Sciences
Dekenstraat 2, 3000 Leuven (Belgium)

## ABSTRACT

Due to its ability to check a decision specification for completeness, consistency and correctness, the decision table technique has always been recognized as a very inviting formalism. Also, the decision table is not an isolated technique and shows a lot of interfaces to other representation formalisms. Few attention, however, has been paid to the modelling process itself and the introduction of computer supported modelling and interfacing of decision tables.

In this paper it is shown how a decision table engineering workbench will create a significant added value to the decision table technique and address these issues of decision table modelling and interfacing.

**Categories and Subject Descriptors:** D.2.2 [**Software Engineering**]: Tools and Techniques-*Decision Tables*; D.2.10 [**Software Engineering**]: Design-*Methodologies, Representation*; H.1.2 [**Information Systems**]: User/Machine Systems-*Human factors*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods-*Representations (procedural and rule-based)*; I.2.5 [**Artificial Intelligence**]: Programming Languages and Software-*Expert system tools and techniques*; I.2.6 [**Artificial Intelligence**]: Learning-*Knowledge Acquisition*; K.6.1 [**Management of Computing and Information Systems**]: Project and People Management-*Systems analysis and Design*

**General Terms:** Human Factors, Verification

## INTRODUCTION

The decision table technique has been recognized as a very inviting formalism for a variety of areas such as program structuring, manual decision making, systems analysis and design, representation of complex texts, verification and validation of knowledge bases, knowledge acquisition. The ability of the decision table to check a given specification for completeness, consistency and correctness has sufficiently been stressed, but few attention has been paid to the modelling process itself and the introduction of computer supported modelling.

Also, the decision table formalism is not an isolated technique and shows a lot of interfaces to other representation formalisms such as program code, trees, rules, etc. Making good use of these connections, however, is only possible through flexible computer support.

For these reasons, a decision table engineering workbench will create a significant added value to the decision table technique. In this paper it is shown how such a workbench, PROLOGA (Procedural Logic Analyzer), addresses these issues of decision table modelling and interfacing. The PROLOGA system (VANTHIENEN [20]) is a PC based interactive design tool for computer supported construction, manipulation, validation and optimization of decision tables.

## THE ADDED VALUE OF AUTOMATED DECISION TABLE CONSTRUCTION

Originally, decision tables were used as an unambiguous and surveyable representation of a set of computer instructions. Above all the ability of the decision table to represent complex decision situations in a clear, well-structured manner was identified as a major advantage : it allows to check the given specifications for completeness, consistency and correctness in a fast and simple way.

The first interest was to convert the decision table into program code, leading to commercial preprocessors and directing research efforts towards efficient conversion algorithms (both in memory usage and execution time). This emphasis on the conversion process, however, restricted the focus of attention in decision table research and practice to a limited area, thereby leaving some important questions unanswered :

- *How are decision tables constructed ?*, and still more important :
- *How to introduce computer support in the decision table construction effort ?*

Gradually more attention was paid to the first question : the design of methods for decision table construction (VERHELST [24,26], CODASYL [3]). With the application field of the decision table shifting away from programming and expanding into other areas of complexity, the conversion problem became less relevant, but the availability of good construction methods was vital to producing high quality decision tables.

The second item, however : the introduction of the computer in decision table construction received very few attention. Some interesting developments were seen (e.g. VIEWEG [27], JOHNSON & KING [7], WELLAND [28]), but in most cases the role of the system was limited to checking (or converting) ready made decision tables and little or no support was available for the modelling process itself.

The concept of computer supported modelling (and not just verification) was introduced by VERHELST [26]. The remainder of this paper is based on these theoretical concepts (VERHELST [26], MAES [11], VANTHIENEN [20, 21]) and practical experiences with earlier implementations (MAES [11], CLEMENT & STROOBANTS [2], MAES, VANTHIENEN & VERHELST [13, 14]).

## BASIC DEFINITIONS

A decision table is a tabular representation of a decision situation, where the different combinations of alternative condition states and the resulting decisions are represented as columns in a table. Over the years the ability of decision tables to represent complex decision situations and the possibilities for easy checking of completeness and consistency have been stressed. Although there has been a major change in use and application possibilities, decision tables still look almost the same as twenty years ago.

A decision table consists of four parts :

- The conditions variables are the criteria which are relevant to the decision making process. They represent the items about which information is needed to take the right decision.
- The condition states are the values the conditions can take : every condition has its set of condition states.
- The actions describe the result of a decision.
- The action values are the possible values for a given action.

These four parts are defined more formally :

- $CS = \{CS_i\}$ (i=1..cnum) is the set of condition subjects;
- $CD = \{CD_i\}$ (i=1..cnum) is the set of condition domains,
  with $CD_i$ the domain of condition i, i.e. the set of all possible values of condition subject $CS_i$;
- $CT = \{CT_i\}$ (i=1..cnum) is the set of condition state sets,
  where $CT_i=\{S_{i,k}\}$ (k=1..$n_i$) is an ordered set of $n_i$ condition states $S_{i,k}$. Each condition state $S_{i,k}$ is a logical expression concerning the elements of $CD_i$, such that $\{S_{i,k}\}$ constitutes a partition of $CD_i$;

- $AS = \{AS_j\}$ (j=1..anum) is the set of action subjects;
- $AV = \{AV_j\}$ (j=1..anum) is the set of action value sets,
  with $AV_j$ = {true, false, nil} the set of action values, which is, in first instance, equal for every action subject, for reasons of consistency checking.

The decision table DT is a mapping between the condition states and the action values, such that every condition combination is mapped onto one and only one action configuration. This is :

$DT : CT_1 \times CT_2 \times ... \times CT_n \rightarrow AV_1 \times AV_2 \times ... \times AV_m$ is a function

A decision table is represented graphically (figure 1) : condition and action subjects on the left in the upper or lower part of the table respectively; condition states and action values at the right hand side.

| | | | | |
|---|---|---|---|---|
| $CS_1$ | $S_{1,k}$ | | | ... |
| $CS_i$ | $S_{i,1}$ | $S_{i,m}$ | ... | ... |
| $AS_j$ ... | $a_x \in AV_j$ ... | $a_y \in AV_j$ ... | ... ... | ... ... |

*Figure 1 : decision table representation*

Every column in the decision table represents a decision rule of the form

IF $CS_1$ is $S_{1,k}$ AND $CS_2$ is $S_{2,m}$ AND ...
THEN action $AS_j$ AND ...

Different concepts of decision tables exist. Mutual exclusivity of columns is the most important criterion to establish a taxonomy. We discern :

1. *multiple hit tables* (tables with non-exclusive columns)

   a) *first hit*: If more than one column is applicable in a certain situation, the first applicable column, from left to right, defines the set of actions to be undertaken.

   b) *all hits* : All applicable columns are taken into account to determine the set of actions to execute. This kind of table can be the first result in the knowledge acquisition phase and can later on be transformed into a single hit decision table.

2. *single hit tables* (decision tables)

   a) *expanded :* Every possible single combination of condition states is represented as a separate column. This representation is used to check whether the table is correct and completely filled out.

   b) *contracted* : Combinations of adjacent (groups of) condition states which lead to the same set of actions are contracted. This leads to a more compact representation.

Only **contracted single hit tables** offer the full advantages of the decision table concept. This however does not imply that the other forms are useless. The multiple hit (all hits) table primarily serves a specification function, whereas the expanded table is best suited for detailed verification purposes. In the decision table construction workbench both forms are used in this order, as intermediate steps in the process leading to the final contracted table.

## BASIC MODELLING FUNCTIONS OF THE WORKBENCH

The PROLOGA (PROcedural Logic Analyzer) system is a PC based interactive design tool for computer supported construction and manipulation of decision tables. The system not only supports the manual design techniques, but also offers additional features to enhance construction, manipulation, validation and optimization of decision tables.

This paragraph describes the facilities PROLOGA offers for the construction and manipulation of decision tables. First a short overview of the manual construction method is given.

### Manual construction of the decision table

Different methods for constructing decision tables can be distinguished (see VERHELST [26]). They essentially contain similar phases for problem analysis, although not always in the same order. The following steps have to be undertaken :

1. Define the conditions, the condition states and the actions.
   - 1.1. Draw up the list of all condition statements and actions that are mentioned in the specification.
   - 1.2. Delete the restatements and the complements from this list.
   - 1.3. Bring together the condition statements that are related to one condition subject such that an exhaustive set of mutual disjoint states is obtained for that condition.
   - 1.4. Fill out the names of conditions and actions in the stub of the table. (Choose any natural order for the actions and conditions, taking into account possible order restrictions.)

2. Define the specification rules

During this phase, the problem is described as a series of logical IF ... THEN ... relations where the connection is made between a combination of condition states and some actions that must be executed.

   - 2.1. Conceive the problem situation (without interpretation).
   - 2.2. Determine the impossible condition combinations and the other relations between the conditions (or actions).
   - 2.3. Describe the problem using logical expressions.

3. Fill out the decision table.

A distinction can be made between filling out the action part and the condition part of the table :

3.1. Fill out the condition entries of the table in lexicographical order, i.e. the lower conditions will vary first. (This can already be performed after step 1.4.)

3.2. Indicate all impossibilities. Impossibilities can be represented in the table in several ways, depending on the preferred option.

3.3. Fill out the action entries (column by column, action by action or rule by rule).

4. Check for completeness, correctness and consistency

4.1. Examine the empty columns. These columns should be examined one by one to verify if it really was the intention to have no actions.

4.2. Examine the unreferenced actions (or conditions).

4.3. Examine the completeness of actions and columns. Some groups of actions should have at least one occurrence. This is usually the case if the actions are the representation of an "extended-entry" action (exhaustivity requirement).

4.4. Examine the table for contradictions. Some actions or action groups may exclude each other and therefore cause contradictions if they occur in the same column (exclusivity requirement).

4.5. Examine the table for correctness. Here we should not only check if the different columns correspond with the given specifications, but also if this specification corresponds with the desired reality.

5. Simplify the decision table

5.1. Contraction of the table. Adjacent (groups of) columns with the same action configuration are contracted into combined columns (e.g. indicating irrelevant condition states). Row order is preserved here.

5.2. Decide upon a suitable layout. To achieve this, several actions can be combined into extended entry actions, as long as this does not cause any contradictions.

6. Optimize the decision table

6.1. Row order optimization. Changing the order of the conditions might result in a contracted decision table with fewer columns.

6.2. Depending on the purpose of the decision table, it might be transformed into optimal test sequences, taking into account condition test times and column frequencies.

### The rationale behind PROLOGA

From the previous list of steps it is obvious that a major drawback of the use of decision tables is the complexity of the manual building process. A lot of redrawing work results from small changes like adding a condition, a condition state or an action. Some manipulations like the reordering of conditions are quite impossible to perform manually. It is a major aim of PROLOGA to free the decision table developer from this cumbersome **drawing** job (VANTHIENEN [22]).

In addition, however, PROLOGA was designed to offer some fundamental **modelling** issues :

- A powerful specification language allows the designer to formulate the decision specification in a straightforward way. In the language provisions are made for expressing general rules, exceptions, preliminary results, restrictive causes and consequences, etc.

- A number of routine jobs like filling action entries from decision rules, generating all condition combinations (without any missing combinations) can be done in a faster and more correct way by the computer.

- The modelling process can be simplified considerably by the use of interactive possibilities such as automatic checking for consistency, correctness and completeness or recommendations for a specific construction method.

- The system can be used for optimization purposes, such as optimal contraction, layout, decomposition into subtables or conversion into efficient program code.

In the following paragraph a description of the basic modelling process is given. More advanced features of PROLOGA will be described in a later section.

### Basic features of the automated construction process

In PROLOGA the construction process largely follows the same steps as described in the manual construction method.

When building a decision table, the designer essentially provides the system with the following information : a list of conditions with their states, a list of actions and a list of relations between condition states and actions (in the form of logical expressions or rules). This will enable the system to construct, display and optimize the corresponding decision table.

Step 1 : Define the conditions, condition states and actions (figure 2).

When conditions are defined, a list of condition states is expected for each condition. The decision table is not restricted to the traditional limited entry table.

Actions have a standard set of action values, only the action subjects must be defined. The possible action values are :

- x : this action should be executed.
- \- : this action must not be executed
- . : undefined
- ? : contradiction (in the resulting decision table)

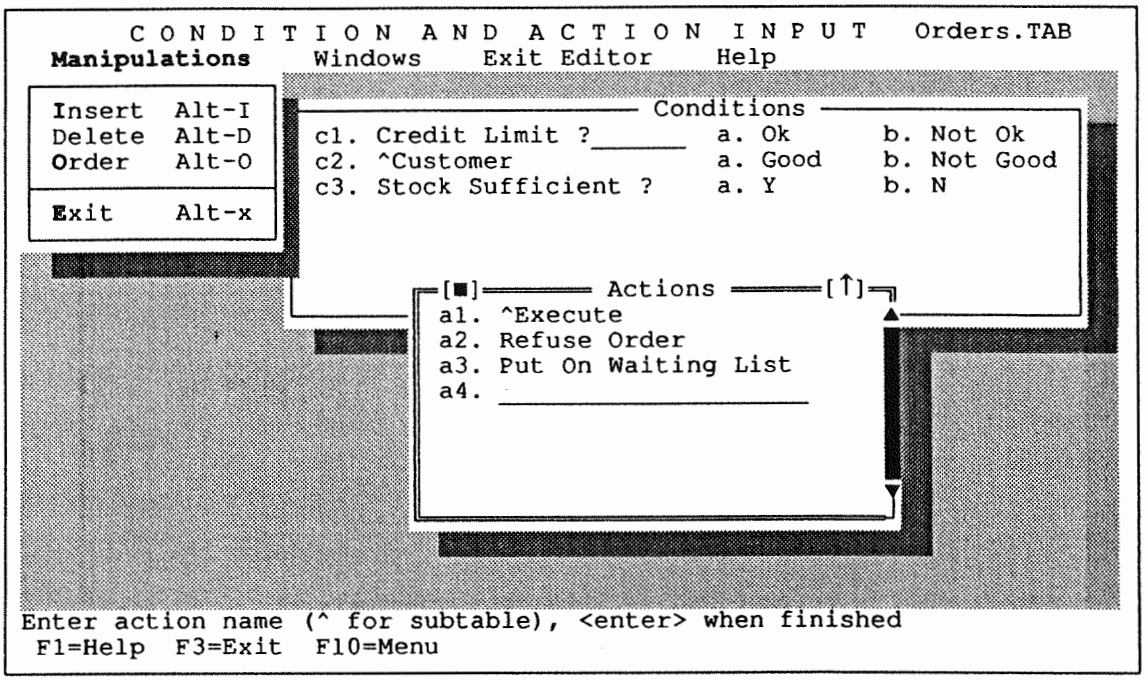


*Figure 2 : condition and action definition*

Step 2 : Define the rules.

For the definition of the decision rules PROLOGA has its own specification language.

To resemble closely the decision situation that has to be modelled, the specification language offers more and also more powerful facilities than simple IF-THEN rules with AND/OR operators. Designing the procedural decision situation requires a specification language which closely matches natural language and its nuances. A decision situation usually does not consist of a collection of independent descriptions, but contains several levels of structure, e.g. general rules, exceptions, ... (MAES [11], MAES & VAN DIJK [12]).

Logical expressions can therefore be assigned different levels of significance, in the sense that certain expressions (general rules) can be overruled by later specifications (exceptions), or that an expression can not be neutralized. Basically two levels are distinguished : **definite** and **preliminary** consequence. In combination with the ONLY operator a **possible** consequence is also provided.

A decision rule then consists of three parts : an action part, an IF part and a condition part. Some example skeletons of decision rules :

*Actions **[generally] if** condition combinations*
***Not** action **definitely if** condition combinations*
*Action **only possible if** condition combinations*
*Action **definitely if and only if** condition combination*

Condition combinations are expressed using several logical operators : not, and, nand, minus, or, xor, nor, ... For a more detailed discussion of the decision rules, see VANTHIENEN [20].

Step 3 : The decision table is filled out

Every decision rule, once defined, is automatically added to the decision table. The decision table is available for display at any time, in expanded or contracted form (and also in the form of a horizontal or vertical decision tree).

Step 4 : Check for Completeness, correctness and consistency

Before the decision table is displayed (figure 3), it is checked for :

- empty columns
- unreferenced conditions or actions
- contradictions

A more detailed description of the knowledge validation possibilities PROLOGA offers, is given in a later section.

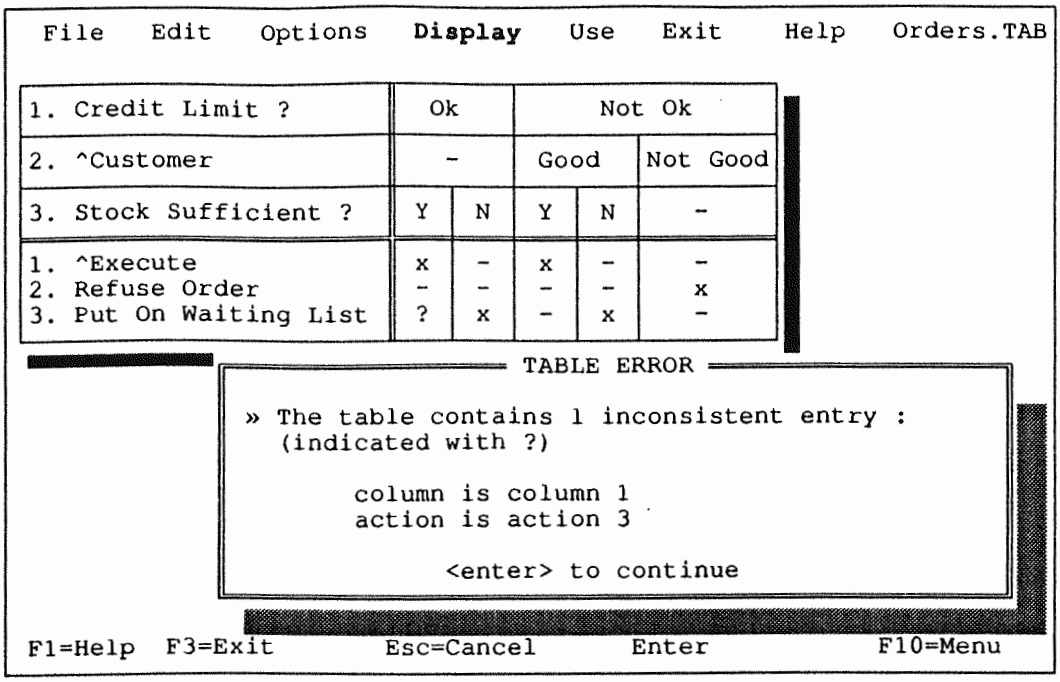


*Figure 3 : checking for consistency*

Step 5 : Simplify the decision Table

From the expanded decision table, adjacent columns or groups of columns leading to similar action configurations are contracted into combined columns, thereby minimizing the number of columns for the given condition order. When all the states of a condition can be combined this will lead to a condition state which is irrelevant, but partial contraction is also provided.

Step 6 : Optimize the decision table

When the decision table has been constructed and verified, it can be subject to different optimizations, e.g. :

- *Row order optimization* : This determines the condition order which results in the minimum number of (contracted) columns. The condition order is the same for all columns of the decision table. For a table with $n$ conditions, this implies a choice between $n!$ alternative condition orders, some of which might be infeasible because of precedence constraints.
- *Execution time optimization* : Depending on the purpose of the decision table, it might be transformed into optimal test sequences, taking into account condition test times and column frequencies (if available). In the resulting execution tree, conditions are not always tested in the same order anymore. For a table with $n$ limited entry conditions, this implies a choice between $f(n)$ decision trees, where :

$$f(n) = n.[f(n-1)]^2, \text{ with } f(1) = 1$$
$$= n.(n-1)^2.(n-2)^4.(n-3)^8..2^{2^{**}(n-2)}$$

$$= \prod_{i=0}^{n-1} (n\text{-}i)^{2^i}$$

$$= \prod_{j=1}^{n} j^{2^{(n-j)}}$$

Throughout the modelling effort, the following features are available in order to enable a flexible construction and manipulation of the decision table(s) :

- inserting, deleting and changing conditions, condition states or actions. The updates are immediately reflected in the decision table.
- changing the order of conditions, condition states or actions.
- reference to condition or action subtables (indicated with '^') and display of the hierarchy of decision tables.
- changing the table layout : expanded versus contracted, minimal column width, state repetition, column numbers, etc.
- display and consultation of the equivalent decision tree (Walk function).
- transformation of the table to other formalisms : code generation, optimal decision tree, consultation monitor, expert system shells.

## ADVANCED FEATURES OF THE DECISION TABLE WORKBENCH

As mentioned in previous sections, a major advantage of automated decision table construction is in the possibilities it offers through links with other knowledge representation formalisms. Therefore a variety of bridges have been built between PROLOGA and other representations, resulting in a large application domain for decision table modelling. The next figure gives an overview of these bridges.

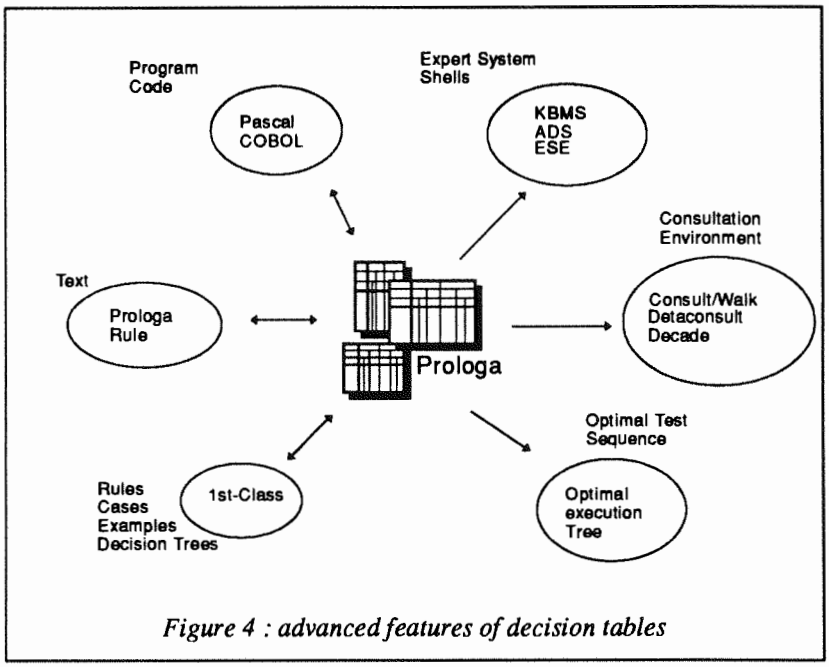


*Figure 4 : advanced features of decision tables*

### Text

Building a decision table from a text is a common application area, where the overview and communication abilities of the visual two-dimensional representation are highly appreciated. Texts are often action oriented and therefore not suited for decision making, which is essentially condition oriented. Examples are : legal texts, management procedures, systems analysis and design, or briefly, descriptions of procedural decisions. To facilitate the transformation process, a powerful specification language has been developed for PROLOGA. Starting from declarative rules expressed in this language, PROLOGA automatically fills in the decision tables (VANTHIENEN [20]).

The opposite direction (from tables to text), however, is also interesting : rebuilding a text or a set of action oriented rules from a correct decision table offers new perspectives in such areas as education, translation, legislation, etc. This rewriting feature is currently being investigated in the context of PROLOGA.

### Code Generation and Verification

In some cases decision tables are designed to be converted to executable program code. This conversion can be done manually or using a preprocessor. In a lot of cases it will be useful to determine optimal test sequences when translating the decision table into program code, thereby minimizing program execution time. A lot of research about decision tables has been going in this direction (CODASYL [3], WELLAND [28]). But even if the decision table is simply converted from left to right and from top to bottom in nested if-then-else structures (without further optimization), usability of code is quite high, especially for prototyping and testing purposes or in cases where minimizing execution time is not worthwhile. PROLOGA can handle both types of conversion.

Currently two languages are supported : Pascal and COBOL. The program code generator will produce the COBOL EVALUATE statement or the structure of the resulting *if-then-else*-statement (according to PASCAL syntax).

In the opposite direction, PROLOGA is able to construct a decision table from existing program code, e.g. from a COBOL EVALUATE statement. As the EVALUATE statement is a multiple hit decision structure, PROLOGA will transform it to a single hit decision table, allowing the designer to check the statement for consistency and completeness.

### Expert System Shells

Decision situations are characterized by three aspects : knowledge acquisition, knowledge validation and maintenance, and decision making. A lot of current knowledge base systems, however, being strong in knowledge representation and decision making, offer little or no guarantees to support structuring, validation, and change control. Validation and verification of the knowledge base proves to be one of the main problem areas in designing these systems and is receiving increased attention (LOVELAND & VALTORTA [10], O'LEARY, GOUL et al. [17]).

Structuring the knowledge in a large number of rules, designed independently, may lead to such problems as incorrectness, inconsistency, incompleteness and redundancy (NGUYEN [16], AYEL [1], LIU & DILLON [9]) :

| | |
|---|---|
| inconsistent and incorrect knowledge : | conflicting rules, |
| | cyclical rules, |
| | invalid attribute values, |
| | unreachable conclusions, |

| | |
|---|---|
| redundant knowledge : | subsumption, |
| | redundant premisses, |
| | redundant rules, |
| incomplete knowledge : | missing knowledge, |
| | unused attribute values or combinations, |
| | unreachable conclusions. |

Most of these common verification problems in rule based systems can easily be solved using decision tables (see e.g. VANTHIENEN [20, 23], CRAGUN & STEUDEL [5], PUURONEN [18], COLOMB & CHUNG [4]). Detecting these shortcomings, either by the designer or automatically, without some form of decision tables, is highly improbable, unless through extensive testing.

The opposite direction (from decision tables to shells), however, is our main concern, because in addition to verification, also the knowledge acquisition process is well served through the overview and communication abilities of well structured decision tables (MERLEVEDE & VANTHIENEN [15]). These table can be converted to a full knowledge based application with advanced consultation possibilities, by linking the decision table workbench to expert system shells. In that way the knowledge is structured using decision tables and is independent from the consultation interface or tool. Maintaining the knowledge based system is then easier because it can be performed on the original tables followed by a new (partial) generation step. Also the consultation interface can be updated without interfering with the decision logic. Explanations and messages are in a separate file which is linked by the generator. Again the decision logic is independent from the (language of the) explanation file.

The generation process has been implemented for two commercial tools : AionDS and KBMS (from former AION and AICORP companies respectively) and has been applied to real world applications.

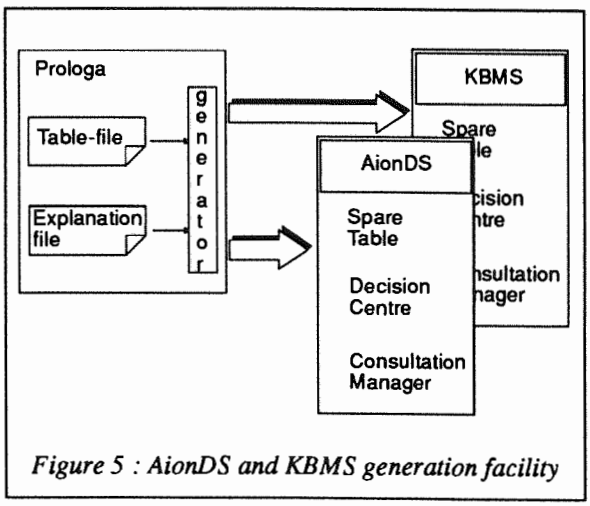


*Figure 5 : AionDS and KBMS generation facility*

On top of the generation of the basic knowledge structure (decision centre) contained in the decision table, PROLOGA can also generate a consultation manager.

The consultation manager provides the application with a question and answer user interface, explanation facilities and the following extra features :

1. *Footsteps* give a chronological survey of the questions asked, the given answer and the conclusion reached by the expert system.
2. *View* gives a list of all condition variables with their values.
3. *What-if mode* and Reconsider offer the possibility to alter previous answers and restart reasoning from the first new answer, thereby allowing what-if simulations.

A Spare Table option offers the possibility to replace the current decision table knowledge in the decision centre by its updated version.

## Consultation Environments

*DECADE/DETACONSULT*

As pointed out previously, code generation to AionDS or KBMS results in a lot of consultation possibilities for decision tables. But PROLOGA also has its own consultation environment : DECADE/DETACONSULT. This decision table consultation environment is similar to an expert system shell, and incorporates rule based knowledge representation, table based verification, extensive consultation facilities (explanation, specific help, selective restart, case archivation) and interfaces to existing procedures and databases (HAZEVOETS, VANHOUTTE & VANTHIENEN [6]).

During a consultation, the user is asked to select the appropriate condition states to determine a unique path through the decision table(s). Only relevant conditions are presented, in the order in which they are included in the decision tables. The resulting actions are then shown to the user or interpreted. Switching from one table to another in a hierarchy of decision tables is performed entirely by the system and is transparent to the user.

Some features are provided which help the user to navigate through the application. For instance, change an answer that was previously entered (all other selections remaining equal) or restart decision making from a certain condition onward. Together with the case archivation option this enables one to perform 'what-if' analyses.

Additional functions include : help facilities, quick resume of the conclusions, links to databases for retrieving or storing information. Finally, there exists an option to save a consultation for later use. This also permits the use of standard cases, where only a few modifications are needed to complete a consultation.

*Consult/Walk*

For quick consultation and prototyping purposes PROLOGA has been provided with at Consult/Walk option. In this way a complete decision situation can easily be consulted (or visualized in walking down the decision tree) via questions and answers, with subtables being activated automatically, but without the features of a full blown consultation environment.

**Optimal Test Sequences**

This option generates optimal test sequences in order to minimize execution time of the decision process, dealing with condition test times and column frequencies..

In decision table research, a lot of effort has (successfully) been spent on generating this optimal tree (see e.g. VERHELST [25], LEW [8], SETHI & CHATTERJEE [19]). Generating optimal test sequences is very useful when the decision table has to be executed frequently or automatically. But generating the optimal tree may be subject to some inconveniences :

1. Determination of the testing cost of a condition;
2. Determination of column frequencies;
3. Performance of the conversion algorithm.

Depending on the application area, execution efficiency is not always the main concern. But even in manual (e.g. administrative) applications, minimizing the number of questions asked (or the number of forms to fill in) might be an important issue.

**1st-Class**

The 1st-Class system (from AICORP) is an inductive learning tool that represents knowledge in the form of trees, optimized programs or rules, starting from a set of examples. The induction algorithm used by 1st-Class is Quinlan's ID3. Looking at a decision table column by column, this table can be seen as a set of examples. In this way 1st-Class can be used to transform and optimize the table in several ways (e.g. conversion to rules, trees, programs).

Other algorithms build an optimal tree starting from a decision table, (e.g. LEW [8]) instead of starting from a set of examples. Part of the research concerning PROLOGA examines the similarities between the inductive approach and approaches generating this optimal execution tree. Both approaches are incorporated in PROLOGA.

The interface between PROLOGA and 1st-Class is a two way interface. The first part of this interface converts a PROLOGA file (which contains the decision table definitions) into a 1st-Class file (which contains the definitions of factors, results and examples).

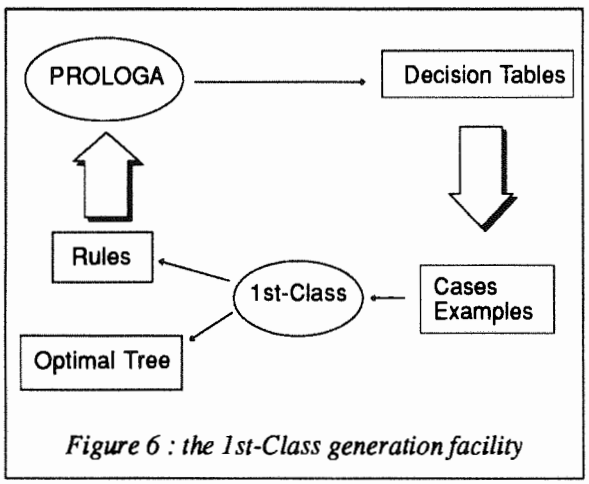


*Figure 6 : the 1st-Class generation facility*

An interesting feature of the use of a decision table as a training set is that one can be sure that the set of examples is complete and consistent.

The reverse part of the interface allows to build a decision table starting from a 1st-Class application.

This is mainly done for two reasons :

- to check 1st-Class applications for completeness and consistency.
- to build decision tables automatically from a (large) set of examples through the inductive learning facilities.

## EXPERIENCES

The decision table engineering workbench PROLOGA has been used in a large number of applications and environments. Some examples of the more common areas of experience include :

- *modelling and verification of complex procedural decision situations in general*;
- *information systems analysis and descriptions of systems requirements*;
- *calculation of rates and premiums in banks and insurance companies*;
- *checking technical specifications in manufacturing*;
- *information systems design and programming*;
- *dialogue structuring facilities and input screen processing*;
- *generating test cases for program structures*;
- *verification and visualization of legal procedures*;
- *legal knowledge based systems*;
- *checking consistency in medical treatments*;
- *help desk applications for computer networks*;
- *validation of knowledge based systems*;
- *knowledge acquisition for medical expert systems*;

The ability of decision tables to represent complex decision situations with easy checking for completeness and consistency makes it a very inviting technique. Besides these classical arguments for the use of decision tables, a number of less known advantages have been revealed by various experiences :

- decision tables act as a *bridge* between modular and partial decision rule specifications (as in expert systems) and efficient condition oriented execution (as in classical programs).

- decision tables offer a uniform technique for the whole *lifecycle* of a decision system, going from the specification phase down to the implementation and maintenance phase, and this for a variety of application areas (management, law, procedures, ...).

- Decision tables easily allow the designers to focus on specific situations (columns) and therefore act as a good medium of *communication* when negotiating, discussing or checking a complex decision situation. Wild discussions about specific exceptions, criteria, cases can immediately be brought down to the real world of condition combinations and columns.

- Even when the knowledge engineer has very limited knowledge of the problem domain (e.g. in medical or legal applications), as an *interviewer* he is able to ask relevant

questions, compare patterns, come up with exceptions, etc. because of the clear representation of the situation.

- Experience has shown that the user or developer who is not familiar with the technique *adapts very well* to the decision table formalism (and sometimes even gets overenthusiastic ...).

- Decision tables can be used in *two directions* : data-driven (forward), determining which actions to undertake in a given situation, and goal directed (backward), specifying under which conditions an action should be undertaken.

- Because the decision logic is represented by the table structure, *translation* from one language to another is easy and only implies translating some terms, not complex logical formulations. In multilingual communities (e.g. the EEC) this constitutes a large advantage.

## CONCLUSIONS

Over the years there has been a major change in research and application areas of decision tables. Decision tables are not an isolated concept with very limited applicability. The renewed interest in this technique in e.g. the area of verification and validation of expert systems illustrates the importance of interfaces to other knowledge representation formalisms.

The features of decision tables can only be exploited in an optimal way if suitable computer support is offered. The decision table workbench PROLOGA presented is this paper is an open tool for the construction and advanced manipulation of decision tables. Besides computer aided modelling, it offers a variety of interfaces to other representation formalisms. This automation of the development process can make the decision table play a central role in designing, validating and implementing a large class of decision situations.

## ACKNOWLEDGEMENT

The authors wish to express their gratitude towards current and former participants in decision table research at K.U.Leuven. This research or paper would not have existed without the presence of M. VERHELST.

## REFERENCES


[1] Ayel, M., A Conceptual Model for Consistency of Knowledge Bases, in : O'Shea, T & Sgurev, V. (Ed.), *Artificial Intelligence III: Methodology, Systems, Applications*, North-Holland, 1988, pp. 75-82.

[2] Clement, P., Stroobants, W., *PRODEMO*, Scientific Report, I.C.M. Paper, K.U.Leuven, Dept. of Applied Economic Sciences, 1982, 75 pp.

[3] Codasyl, A Modern Appraisal of Decision Tables, *Report of the Decision Table Task Group*, ACM, New York, 1982, pp. 230-232.

[4] Colomb, R., Chung, C., Very Fast Decision Table Execution of propositional Expert Systems, *Proceedings AAAI90*, 1990, pp. 671-676.

[5] Cragun, B., Steudel, H., A Decision-Table Based processor for Checking Completeness and Consistency in Rule-Based Expert Systems, *International Journal of Man-Machine Studies*, 1987, pp. 633-648.

[6] Hazevoets, F., Vanhoutte, B., Vanthienen, J., An Expert System Interface for Consultation of Decision Table Systems, *International Conference on Data Base and Expert Systems Applications*, Aug. 29-31, Vienna, 1990.

[7] Johnson, R., King, P., Computer Assisted Decision Table Development, *Computer Bulletin*, 2(15), 1978, pp. 22-24.

[8] Lew, A., Optimal Conversion of Extended-Entry Decision Tables with General Cost Criteria, *Communications of the ACM*, 21(4), April 1978, pp. 269-279.

[9] Liu, N., Dillon, T., Detecting of Consistency and Completeness in Expert Systems using Numerical Petri Nets, in : Gero, J. & Stanton, R. (Ed.), *Artificial Intelligence Developments and Applications*, North-Holland, 1988, pp. 119-134.

[10] Loveland, D., Valtorta, M., Detecting Ambiguity : An Example in Knowledge Evaluation, *Proceedings Eighth International Joint Conference on Artificial Intelligence*, Vol. 1, (August, Karlsruhe), 1983, pp. 182-184.

[11] Maes, R., *Bijdrage tot een Kritische Herwaardering van de Beslissingstabellentechniek*, Doctoral Dissertation, K.U.Leuven, 1981, 397 pp.

[12] Maes, R., Van Dijk, J., *A User-friendly Propositional Formalism for a Managerial DSS-generator*, in : PAU, L. (Ed.), Artificial Intelligence in Economics and Management, North-Holland, 1986, pp. 65-76.

[13] Maes, R., Vanthienen, J., Verhelst, M., Procedural Decision Support Through the Use of PRODEMO, *Proceedings Second International Conference on Information Systems*, Cambridge (Mass.), December 7-9, 1981, pp. 135-152.

[14] Maes, R., Vanthienen, J., Verhelst, M., Practical Experiences with the PROcedural DEcision MOdeling System, *Proceedings Joint IFIP WG 8.3/IIASA Working Conference on Processes and Tools for Decision Support*, (July, 19-21, Laxenburg, Austria), 1982, pp. 139-154.

[15] Merlevede, P., Vanthienen, J., A Structured Approach to Formalization and Validation of Knowledge, *IEEE/ACM International Conference on Developing and Managing Expert System Programs*, Washington, DC, 30/9-2/10/91, pp. 149-158.

[16] Nguyen, T., Perkins, W., Laffey, T., Pecora, D. , Knowledge Base Verification, *AI Magazine*, 8(2), 1987, pp. 69-75.

[17] O'Leary, T., Goul, M. et al., Validating Expert Systems, *IEEE Expert*, june 1990, pp. 51-58.

[18] Puuronen, S., A Tabular Rule Checking Method, *Proc. Avignon87*, Vol. 1, 1987, pp. 257-268.

[19] Sethi, I., Chatterjee, B., Conversion of Decision Tables to efficient Sequential Testing Procedures, *Communications of the ACM*, 23(5), May 1980, pp. 279-285.

[20] Vanthienen, J. , *Automatiseringsaspecten van de specificatie, constructie en manipulatie van beslissingstabellen*, Doctoral Dissertation, K.U.Leuven, Dept. Applied Econ., 1986, 378 pp.

[21] Vanthienen, J., Een moderne kijk op beslissingstabellen, *Informatie*, December 1988, pp. 912-937.

[22] Vanthienen, J., A Longest Path Algorithm to Display Decision Tables, *The Computer Journal*, 34(4), 1991, pp. 358-360.

[23] Vanthienen, J., Knowledge Acquisition and Validation Using a Decision Table Engineering Workbench, *Proceedings of the World Congress on Expert systems*, Pergamon Press, Orlando (Florida), Dec. 16-19, 1991, pp. 1861-1868.

[24] Verhelst, M., A Technique for Constructing Decision Tables, *IAG Quarterly Journal (later : Information & Management)*, 2(1), March 1969, pp. 27-36.

[25] Verhelst, M., The Conversion of Limited-Entry Decision Tables to Optimal and Near-Optimal Flowcharts : Two New Algorithms, *Communications of the ACM*, 15(11), Nov. 1972, pp. 974-980.

[26] Verhelst, M., *De Praktijk van Beslissingstabellen*, Kluwer, Deventer/Antwerpen, 1980, 175 pp.

[27] Vieweg, W., *Die Konstruktion von Entscheidungstabelle*, Betriebswirtschaftliche Verlag Dr. Th. Gabler, Wiesbaden, 1973, 232 pp.

[28] Welland, R., *Decision Tables and Computer Programming*, Heyden & Son Ltd., Bury St. Edmunds, Great Britain, 1981, 203 pp.

## CONTENTS


ABSTRACT . . . . . 1
INTRODUCTION . . . . . 2

THE ADDED VALUE OF AUTOMATED DECISION TABLE CONSTRUCTION . . . . . 3

BASIC DEFINITIONS . . . . . 4

BASIC MODELLING FUNCTIONS OF THE WORKBENCH . . . . . 7
Manual construction of the decision table . . . . . 7
The rationale behind Prologa . . . . . 9
Basic features of the automated construction process . . . . . 9

ADVANCED FEATURES OF THE DECISION TABLE WORKBENCH . . . . . 14
Text . . . . . 14
Code Generation and Verification . . . . . 15
Expert System Shells . . . . . 15
Consultation Environments . . . . . 17
DECADE/DetaConsult . . . . . 17
Consult/Walk . . . . . 18
Optimal Test Sequences . . . . . 18
1st-Class . . . . . 18

EXPERIENCES . . . . . 19

CONCLUSIONS . . . . . 22
ACKNOWLEDGEMENT . . . . . 22
REFERENCES . . . . . 23
CONTENTS . . . . . 26